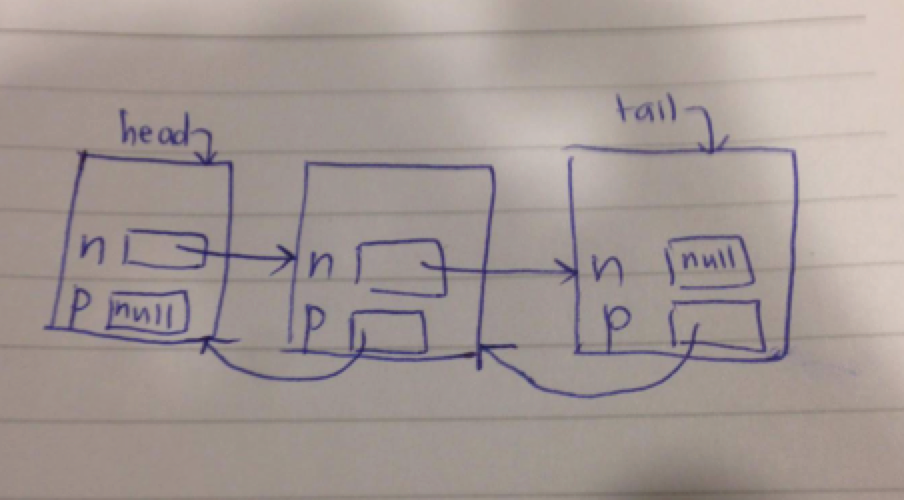
Description of the design of your doubly-linked list implementation.

* Doubly linked list with a head and a tail pointer
* Head->prev points to nullptr, tail->next points to nullptr
* Insert function will put new item into head of linked list
* Typical map:



* Empty map just has head and tail pointers to nullptr

Pseudocode for non-trivial algorithms (e.g., Map::erase and subtract).

**Map::erase**

If map contains key

If map size is 1

Erase key

If key is head key

Erase key

If key is tail key

Erase key

If none of these conditions (i.e. key is in middle of map)

Make temp equal to pointer to key

Erase key

Delete temp pointer

**Combine**

If both m1 and m2 are empty

Empty result map

If m1 is empty but m2 is not empty

Empty result map

Make result map equal to m2

If m2 is empty but m1 is not empy

Empty result map

Make result map equal to m1

Make copies of m1 and m2 (m1Copy and m2Copy)

Loop through m1Copy

Get values for key and value from an item in m1Copy

Loop through m2Copy

Get values for key and value from an item in m2Copy

If key is present in both maps and value is equal

Erase key from m2Copy

Decrement loop number of m2Copy to take erased key into account

If key is present in both maps and value is different

Erase key from both m1Copy and m2Copy

Increment count int variable

Decrement loop number of m1Copy and m2Copy to take erased key into account

Empty result map

Make result map equal to m1Copy

Insert items in m2Copy into result map

If count was incremented (i.e. if there was a key present in both maps with different values)

Return false

Return true

**Subtract**

Make copy of m1 (m1Copy)

Loop through m1Copy

Get values for key and value from an item in m1Copy

Loop through m2

Get values for key and value from an item in m2

If key is present in both maps

Erase key from m1Copy

Decrement loop number of m1Copy to take erased key into account

Empty result map

Insert items in m1Copy into result map

List of test cases that would thoroughly test the functions.

ValueType value = 50;

KeyType key = "before";

Map a, b;

// test insert

assert(a.insert("Adam", 10));

// test empty

assert(!a.empty() && b.empty());

// test size

assert(a.size() == 1);

// test insert for when key is already present

assert(!a.insert("Adam",5));

// test update for when key is present/not present

assert(a.update("Adam", 5) && !a.update("Bob", 10));

// test insertOrUpdate for when key is present/not present

assert(a.insertOrUpdate("Adam", 10) && a.insertOrUpdate("Bob", 15));

assert(a.size() == 2);

// test contains

assert(a.contains("Adam") && a.contains("Bob") && !a.contains("Chad"));

// test erase for when key is present/not present

assert(a.erase("Adam") && !a.erase("Chad"));

assert(a.size() == 1);

assert(a.contains("Bob") && !a.contains("Adam"));

// test get value

assert(a.get("Bob", value) && value == 15 && !a.get("Chad", value));

// test get key and value

assert(!a.get(1, key, value) && a.get(0, key, value) && key == "Bob" && value == 15);

// create new map b to test swap

b.insert("Alice",5);

b.insert("Belinda", 6);

b.insert("Carmen", 7);

b.insert("Diana", 8);

a.swap(b);

assert(a.size() == 4 && b.size() == 1);

assert(a.get("Alice", value) && a.get("Carmen", value) && !a.get("Bob", value) && b.get("Bob", value));

// test combine for return true

Map c, d, result1;

assert(c.insert("Fred", 123) && c.insert("Ethel", 456) && c.insert("Lucy", 789));

assert(d.insert("Lucy", 789) && d.insert("Ricky", 321));

assert(result1.insert("Adam", 1) && result1.insert("Bob", 3));

assert(combine(c, d, result1));

assert(result1.size() == 4);

assert(result1.get("Fred", value) && value == 123);

assert(result1.get("Ricky", value) && value == 321);

assert(result1.get("Lucy", value) && value == 789);

assert(result1.get("Ethel", value) && value == 456);

// test combine if result and m1 is same map

assert(combine(d, c, d));

assert(d.size() == 4);

assert(d.get("Fred", value) && value == 123);

assert(d.get("Ricky", value) && value == 321);

assert(d.get("Lucy", value) && value == 789);

assert(d.get("Ethel", value) && value == 456);

// test combine for return false

Map g, result3;

assert(g.insert("Lucy", 654) && g.insert("Ricky", 321));

assert(!combine(c, g, result3));

assert(result3.size() == 3);

assert(result3.get("Fred", value) && value == 123);

assert(result3.get("Ricky", value) && value == 321);

assert(result3.get("Ethel", value) && value == 456);

// test operator assignment

c = d;

assert(c.size() == 4);

assert(c.get("Fred", value) && value == 123);

assert(c.get("Ricky", value) && value == 321);

assert(c.get("Lucy", value) && value == 789);

assert(c.get("Ethel", value) && value == 456);

// test copy constructor when map being copied is empty

Map j;

assert(j.insert("YY", 5));

Map i = j;

assert(i.contains("YY") && i.size() == 1);

assert(j.erase("YY"));

Map k = j;

assert(k.empty());

// test subtract

Map e, f, result2;

assert(e.insert("Fred", 123) && e.insert("Ethel", 456) && e.insert("Lucy", 789));

assert(f.insert("Lucy", 789) && f.insert("Ricky", 321) && f.insert("Ethel", 654));

assert(result2.insert("Adam", 1));

subtract(e, f, result2);

assert(result2.size() == 1);

assert(result2.get("Fred", value) && value == 123);

cout << "All tests passed." << endl;